Energy Consumption Library

Leandro Fontoura Cupertino, Georges da Costa, Amal Sayah, Jean-Marc Pierson

Energy Consumption Library

Leandro F. Cupertino(✉), Georges Da Costa, Amal Sayah,
and Jean-Marc Pierson

Toulouse Institute of Computer Science Research (IRIT),
University of Toulouse III (Paul Sabatier),
Toulouse, France
{fontoura, dacosta, sayah, pierson}@irit.fr

Abstract. The energy consumption of a computing system depends not only on its architecture, but also on its usage. This paper describes the Energy Consumption Library (`libec`), a modular library of sensors and power estimators, which do not depend on wattmeter to measure the power dissipated by a machine and/or the applications that it executes, etc. In addition, four use cases are used to demonstrate some of the library's capabilities.

1 Introduction

The power dissipated on data centers is highly increasing along time. It is known that the cost of maintaining such servers during two years can be greater than the cost of the hardware itself. The energy fraction spent by Information and Communications Technologies (ICT) over the worldwide available electricity is estimated to double in twelve years [1].

Initially the focus of energy savings on ICT was related to hardware enhancements. However, the power dissipated by a computing system is not static, i.e., it depends not only on its hardware specification but also on its usage. Distinct workloads will waste different amount of energy, which can vary even for the same application running on the same hardware depending, for instance, on its communication issues. The understanding of how the energy is used by an application can be used in software engineering to deploy libraries, implementations or compilation parameters to achieve energy-aware software. This knowledge can also be used on data centers to schedule the resources properly, taking into account the available electrical energy contracts. Several papers proposes different power models for estimating the energy consumption of applications according to its workload [2–5].

In this paper we present the Energy Consumption Library (`libec`), an open source library of sensors that can estimate the power dissipated by a machine or an application even without the presence of a wattmeter on the host machine. The remainder of this paper is divided as follows. Section 2 describes the library and its features. In Sect. 3 we present two use cases for this library, a process monitor and an application profiler. Finally, Sect. 4 draws some conclusions over the developed library.

2 The Library

The main goal of the Energy Consumption Library, `libec`, is to provide a modular library to aid the development of new power estimators. To be easy to extend and maintain, it was implemented in C++ and is distributed under the GNU General Public License (GPL) version 3.0 or later. It can be downloaded from [6]. This library contains a set of sensors as input variables in several power models. The information provided from the sensors comes mainly from Linux kernel's API, `/sys` and `/proc` file systems. Nevertheless, these sensors can be extended in order to collect different data coming from any source specific sensors.

The `libec` sensors can be implemented at two levels: machine and/or application. The application level contains all the sensors that can be directly associated with a process identification (PID), these sensors are mainly related to software usage, such as performance counters. Meanwhile, the machine level has not only the aggregated value for all the processes, but also some physical properties measurements that cannot be associated to a PID, such as the CPU thermal dissipation. Furthermore, there is a special kind of application level sensor which is application's power estimators. The next subsections describe each available sensor.

2.1 Application/Machine Level Sensors

In order to estimate the energy consumed by an application, one need to have at least one application related variable, i.e., a variable which can retrieve application's information. With that in mind, `libec` has some PID related sensors. These sensors can gather not only application, but also machine level information.

The most power consuming devices on servers are CPU, memory and disk. In other words, in order to achieve good power models, one needs to access information regarding the usage of such devices. On the CPU side, one can exploit Performance Counters, CPU time, CPU elapsed time and CPU usage. Furthermore, sensors with memory usage and disk read/write can be used for memory and disk modeling, respectively. The available application's sensors for are implemented as follows.

Performance Counters (PCs) are hardware event counters that use special file descriptors to count them. They are available through the Linux kernel API [7]. This sensor gives the count hits between two updates, which are defined by the user. PCs can provide information related to a CPU, such as clock cycles, instructions, cache references and misses, branch instructions and misses, page faults, context switches, among others. In order to access the performance counters, one needs to have administrative privileges.

In order to enable more flexibility, the *CPU usage* sensor is composed by two other intermediate sensors: *CPU time* and *Elapsed CPU time*. *CPU time* provides the total CPU time, i.e., the sum of the system and user times. This information is retrieved from the `/proc/[PID]/stat` file. For the machine level information, this data is available in the `/proc/stat` file. This sensor can also provide the total elapsed time between updades, i.e., system, user and idle time. The returned time value is provided in clock ticks. The *Elapsed CPU time* sensor uses the information from the *CPU time* sensor to measure the CPU time difference between two updates. *CPU usage* (CPU%) provides the percentage of CPU utilization of a specific PID or CPU core. For the moment it cannot return the utilization of both at the same time, i.e., one cannot request the CPU usage of a PID regarding to one specific core, but only the PID's CPU usage on the entire machine or the core's usage for the machine as a whole. This sensor uses the elapsed CPU time sensor and divides it by the machine level CPU elapsed time, i.e., the total elapsed time.

Memory usage (MEM%) provides the percentage of memory used by a given process. It collects application's resident set size and divide it by the total available memory found in the `/proc/[PID]/stat` and `/proc/meminfo` files.

Disk Read/Write provides the number of read/written bytes between function calls available at the `/proc/[PID]/io` and `/sys/block/[dev]/stat` file. This sensor can retrieve information for any file partition that may come from a flash drives or an IDE hard drive.

2.2 Machine Level Sensors

In addition to the application level sensors, which can also be used to collect machine related information, `libec` contains some other sensors that can only be attributed to the machine as a whole. This section describes the sensors that are only available at the machine level.

The *CPU temperature* sensor retrieves its information from the `/sys` file system, but its file varies according to the CPU vendor. This information is triggered at the constructor of the class.

CPU frequency is also available in the `/sys` file system. In order to read such file, one needs to install the required packages and have administrative privileges. To enable all users to use such sensor, even if the information access is slower, the `/proc/cpuinfo` file is used according to the user's privileges.

The *Networking* traffic information is retrieved from the `/proc/net/dev` file. The user must define if the retrieved data will come from the sum of the networking devices or from just one of them. Besides, user can decide which type of data will be used (packets/bytes received/transmitted).

Some wattmeters interfaces can also be found on libec. To facilitate the library's use not only on server but also on notebooks, there is a meter which collects PDU power directly and another one which exploits the ACPI information to estimate the portable device power. The *ACPI Power Meter* retrieves information related to the voltage and current drained by the battery from the `/sys/class/power_supply` folder and calculates its power consumption. Its drawback is that it requires ACPI enabled hardware. On the server side, the PDU's communication deeply depends on the vendor's protocol used. Due to personal uses, we made available meters for some of the Grid5000's experimental testbed nodes [8], the RECS system [9] and Energy Optimizers Limited's Plogg (an outlet adapter to measure the dissipated power of devices).

2.3 Application's Power Estimators

The main target of this library is to enable users to develop new power estimators. For the moment, one static and two dynamic model where implemented. Static models require a priori information, while dynamic ones can auto adapt to different workloads but must have access to a power meter.

The simplest static model is a CPU proportional model. Our static model, namely *CPU MinMax*, is a linear estimator based on the minimum (P_{min}) and maximum (P_{max}) power consumption of the machine. This information must be provided by the user. It uses a *CPU usage* sensor to weight the power variance and the number of active processes ($|AP|$) to normalize the idle power (P_{min}) for each process as stated in Eq. 1. One must be aware that this estimator is architecture dependent and its performance varies according to the accuracy of the data provided by the user.

$$P_{pid} = (P_{max} - P_{min}) \times CPU\%_{pid} + \frac{P_{min}}{|AP|} \tag{1}$$

When a wattmeter is available, one can exploit it to achieve more precise results or to calibrate their models to use in similar machines that do not have such device. The *Inverse CPU* power estimator uses the information from the total energy consumption of a machine and attributes it to the application level by the use of a *CPU usage* sensor as stated below

$$P_{pid} = P_{machine} \times CPU\%_{pid}. \tag{2}$$

One well known method for achieving dynamic estimators is the use of linear regression method to weight some pre-defined sensors and find a model without user provided information. The *Linear Regression Dynamic Power Estimator* can do so by estimating the weights (w_i) for any application level sensor (s_i) within the follow equation

$$P_{machine} = w_0 + \sum_{i=1}^{n} w_i \times s_i. \tag{3}$$

3 Use Cases

In this section, we present four use cases for the `eclib`. The idea here is not to evaluate the results in a detailed way, but only to show the user, what kind of information it can produce using `libec`. The first use case shows how to implement your own sensor for it to be compatible with the other tools developed with `libec`. The second illustrates an easy way to monitor the top consuming processes running on the machine through the `ectop` tool. The third one is how to profile the energy spend by an application with Valgreen [10]. Finally the importance of a workload adaptative model is illustrated by changing the load of a machine and monitoring different types of power estimators. All of these use cases are available within the `ectools` package [6].

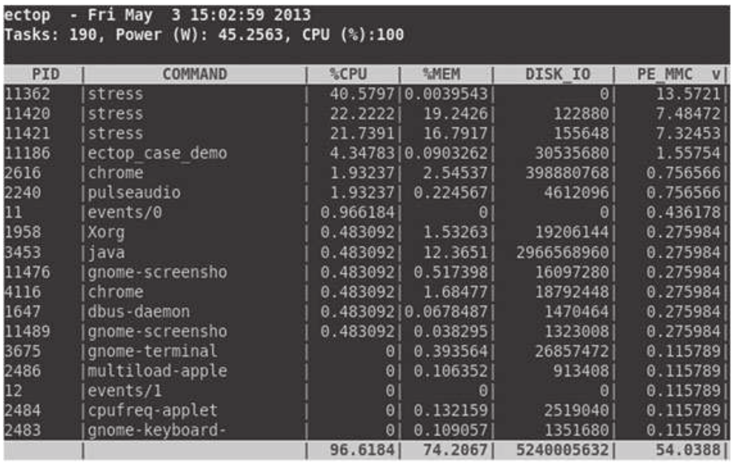
3.1 Extending Sensors

New machine and application sensors can be easily implemented by extending the *Sensor* and *SensorPID* classes, respectively. To do so, the user must at least overload the *update* and *updatePid* methods, if specific data structures are used, it may be necessary to overload the *getValue* and *getValuePid* methods as well.

3.2 Power Monitoring Tool

The Energy Consumption Monitoring Tool (`ectop`) application was conceived to provide an easy way to monitor power estimators and compare their accuracy in real time. It is a command line interface in which the user can keep track of the top consuming processes. It also allows the user to add/remove application level sensors and power estimators. To add/remove a sensor from the interface, one simply needs to instantiate the desired sensor class and add it to the monitor though the *addSensor* method.

Figure 1 shows an example of `ectop` with three sensors (CPU and memory usage and disk I/O) and one power estimator (min-max CPU proportional, PE_MMC). The `ectop` monitor shows a sum bar for each sensor's column, which gives an idea of its system wide values. For the presented scenario the sum of the power estimations is bigger than the value given in the power field. The power field is filled with the ACPI power estimator. This difference occurs because of the quality of the power estimator used. As stated earlier, `eclib` is a library to aid the development of new estimators and, for the moment, the power estimators present on this library are first attempts to generate broader models.



ectop - Fri May 3 15:02:59 2013
Tasks: 190, Power (W): 45.2563, CPU (%):100

PID	COMMAND	%CPU	%MEM	DISK_IO	PE_MMC v
11362	stress	40.5797	0.0039543	0	13.5721
11420	stress	22.2222	19.2426	122880	7.48472
11421	stress	21.7391	16.7917	155648	7.32453
11186	ectop_case_demo	4.34783	0.0903262	30535680	1.55754
2616	chrome	1.93237	2.54537	398880768	0.756566
2240	pulseaudio	1.93237	0.224567	4612096	0.756566
11	events/0	0.966184	0	0	0.436178
1958	Xorg	0.483092	1.53263	19206144	0.275984
3453	java	0.483092	12.3651	2966568960	0.275984
11476	gnome-screensho	0.483092	0.517398	16097280	0.275984
4116	chrome	0.483092	1.68477	18792448	0.275984
1647	dbus-daemon	0.483092	0.0678487	1470464	0.275984
11489	gnome-screensho	0.483092	0.038295	1323008	0.275984
3675	gnome-terminal	0	0.393564	26857472	0.115789
2486	multiload-apple	0	0.106352	913408	0.115789
12	events/1	0	0	0	0.115789
2484	cpufreq-applet	0	0.132159	2519040	0.115789
2483	gnome-keyboard-	0	0.109057	1351680	0.115789
		96.6184	74.2067	5240005632	54.0388

Fig. 1. `ectop` command line interface

3.3 Application's Energy Profiling

For the profiling of applications an application's energy profiler, namely Valgreen [10] is available. It uses `libec` to aid the programmer to implement energy efficient algorithms by sampling the power consumption of a given application in small time steps. These time steps may be configured and the smaller it goes, more precise the energy measurement will be.

3.4 Dynamic Power Estimators

Auto-generated models have been widely used on the field of application's power estimation. This use case will compare a dynamic model that is updated through linear regression in regular time intervals with a simple static model. These dynamic models do not need any input from the user and can be used with different devices. The idea is to show the importance of a dynamic model when the workload on the machine changes and our model is not valid anymore.

Figure 2 presents a comparison between a dynamic model, the actual power measured with a wattmeter and a static model parameterized in another machine. One can see that as time goes by, the adaptive model gets closer to the actual dissipated power. Here we show the total power consumption of the machine, but the same model can also retrieve the power dissipated by each process.

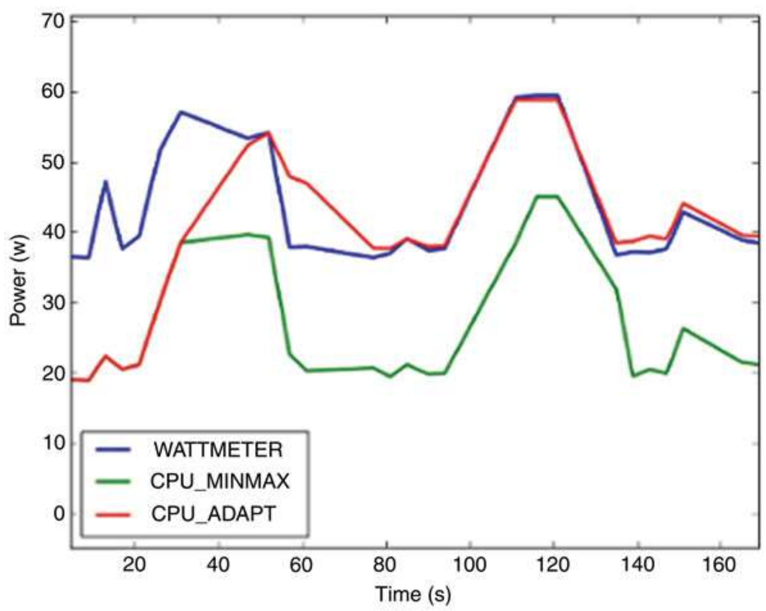


Fig. 2. Comparison between a wattmeter, an adaptive model (CPU_ADAPT) and a static model (CPU_MINMAX).

4 Conclusions

The Energy Consumption Library (`libec`) is a library that can be easily extended and can be used for several different purposes, such as, to compare power estimators in real time, profile applications and efficiently allocate resources taking into account its energy consumption. It is usable on laptops as well as servers and can estimate the power spent by an application even without the presence of a wattmeter.

For the time, `libec` has very few power estimators and it sensors only runs on Linux platforms. As future work we aim to implement new power estimators, as well as expand it to android and windows.

Acknowledgments. The results presented in this paper were funded by the European Commission under contract 288701 through the project CoolEmAll and by the COST (European Cooperation in Science and Technology) framework under Action IC0804.

References

1. Vereecken, W., Van Heddeghem, W., Colle, D., Pickavet, M., Demeester, P.: Overall ICT footprint and green communication technologies. In: 4th International Symposium on Communications, Control and Signal Processing (ISCCSP), pp. 1–6 (March 2010)
2. Rivoire, S., Ranganathan, P., Kozyrakis, C.: A comparison of high-level full-system power models. In: Proceedings of the 2008 Conference on Power Aware Computing and Systems HotPower'08. USENIX Association, Berkele, p. 3 (2008)
3. DaCosta, G., Hlavacs, H.: Methodology of measurement for energy consumption of applications. In: 11th IEEE/ACM International Conference on Grid Computing (GRID), pp. 290–297 (October 2010)
4. Chen, H., Shi, W.: Power measuring and profiling: the state of art. In: Ahmad, I., Ranka, S. (eds.) Handbook of Energy-Aware and Green Computing, pp. 649–674. Chapman and Hall/CRC, Boca Raton (2012)
5. Witkowski, M., Oleksiak, A., Piontek, T., Wglarz, J.: Practical power consumption estimation for real life HPC applications. Future Gener. Comput. Syst. **29**(1), 208–217 (2013)
6. Cupertino, L.F.: Energy consumption tools web-page (www.irit.fr/~Leandro.Fontoura-Cupertino/ectools/) (April 2013)
7. Linux man-pages project: perf_event_open(2) (February 2013)
8. Cappello, F., Caron, E., Dayde, M., Desprez, F., Jegou, Y., Primet, P., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quetier, B., Richard, O.: Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In: The 6th IEEE/ACM International Workshop on Grid Computing, p. 8. IEEE (2005)
9. Christmann: Description for Resource Efficient Computing System (RECS) (2009)
10. Cupertino, L.F., DaCosta, G., Sayah, A., Pierson, J.M.: Valgreen: an application's energy profiler. In: Conference on Soft Computing and Soft Engineering (SCSE) (2013)